Developing Highly Predictable System Behavior in Real-Time Battle-Management Software

Dale Scott Caffall
Missile Defense Agency
7100 Defense Pentagon
Washington, D.C. 20301-7100, USA
butch.caffall@mda.osd.mil

James Bret Michael, Man-Tak Shing
Naval Postgraduate School
833 Dyer Road
Monterey, California 93943-5118, USA
{bmichael, shing}@nps.edu

ABSTRACT

This paper addresses the need for development of highly dependable systems-of-systems for missile defense, which involves C4 systems, which are traditionally non-real-time, and weapon systems, which typically operate under real-time constraints. If we are to match the performance of weapon systems and avoid the negative impact of forcing synchronization of battle manager software with weapon systems for messaging, then we must develop the battle manager as real-time software. We present a new approach for developing battle-management software as a real-time set of system functionality that addresses warfighter usage. To achieve the level of desired predictable battle-management behavior, we maintain that it is essential to develop a formal representation that captures the desired battle manager system behavior and test the formal representation against the expected battle-management properties. Furthermore, we assert that it is critical to develop the battle manager as a real-time software-intensive system to ensure the schedulability of battle-management tasks and provide for concurrent execution of such tasks where applicable.

Keywords: Battle Management Kernel, Missile Defense, System-of-Systems

1. INTRODUCTION

During the past decade, the U.S. Department of Defense (DoD) shifted military tactics from the traditional war of attrition to a transformational concept of full-spectrum dominance: the ability of U.S. forces, operating unilaterally or in combination with multinational and interagency partners, to defeat any adversary and control any situation across the full range of military operations. There is an urgent need for DoD to develop a global missile defense system to protect the U.S. and its allies against hostile missile attack. This paper describes a new approach for developing battle-management software as a real-time set of system functionality that addresses warfighter usage. Section 2 reviews the challenges in missile defense system development. Section 3 presents an overview of the proposed paradigm. Section 4 highlights the formal approaches for developing the battle management kernel. Section 5 discusses the proposed approach and section 6 draws the conclusions.

2. STATEMENT OF THE PROBLEM

In [10], Parnas outlines six major characteristics of the battle-management software in the Strategic Defense Initiative (SDI) (known today as the Ballistic Missile Defense) program; these issues are as relevant today as when the were first raised:

1. The battle-management software must identify, track, and direct weapons towards targets whose characteristics may not be known with certainty until the moment of battle. The battle-management software must discriminate the threat objects from decoys and debris.
2. The battle-management computing will be accomplished through a network of computers that are connected to sensors and weapons as well as other battle-management computers. The behavior of the battle-management software cannot be predicted with confidence given the actual configuration of weapons, sensors, and battle managers at the moment of battle.
3. Developers cannot test the battle-management software under realistic conditions prior to actual use of the software.
4. The duration of the defense engagement will be short: it will not allow for either human intervention or debugging the software to overcome software faults at runtime.
5. The battle-management software will have absolute real-time deadlines for the computation that will consist of periodic processes to include detecting and identifying potential threat missiles, assigning a weapon to engage the threat missile, and providing an assessment of the interceptor-threat missile engagement. Because of the unpredictability of the computational requirements of each process, developers cannot predict the required resources for computation.
6. The missile defense system will include a large variety of sensors, weapons, and battle-management components for which all will be large, complex software systems. The suite of weapons and sensors will increase in number as the development progresses. The characteristics of these future weapons and sensors are not well defined and will likely remain fluid for many years. Additionally, all weapons and sensors will be subject to change independently of each other. As such, the battle-management software must integrate numerous dynamic software systems to the extent that has never before been achieved.

System-of-systems functional and performance expectations of the users continue to increase as the acquisition community continues to develop and field the products of C4 systems and weapon-systems integration. The class of systems in which Command and Control (C2) and battle-management systems are contained is called Command, Control, Communications, and

Computers (C4) systems. Typically, C4 systems are non-real-time systems. Traditionally, weapon systems are real-time systems. If we are to match the performance of the weapon systems and avoid the negative impact of forcing synchronization of the battle manager with the weapon system for messaging, then we must develop the battle manager as real-time software.

As a rule, battle management is still executed at the system level rather than the desired system-of-systems level. We define a system-of-systems as an amalgamation of legacy systems and developing systems that provide an enhanced military capability greater than that of any of the individual systems within the system-of-systems.

Another factor that contributes to the challenge involved in predicting battle-management behavior is the acquisition practices currently employed in DoD. The increased pressure to rapidly move product into the operational battlespace tends to channel program managers into focusing on achieving functionality as quickly as possible. As such, the development community responds with a hurried and oftentimes inadequate design phase and follows with an intense period of coding. In the rush to rapidly develop a product, one can fall into the trap of exclusively seeking some level of achieved capability while ignoring the behavior of the software [2].

The current state-of-the-practice for developing systems-of-systems tends to be *ad hoc* as discussed in [11]. Given that the interconnected battle-management solutions in systems-of-systems are separately designed and developed on different operating platforms, predicting battle-management behavior of the systems-of-systems is not possible; this issue was also raised, in a broader context, in [9].

Because we cannot readily predict the system behavior of legacy battle-management systems, the requirements for such systems tend to be treated as part of a new development. While the basic five functions do not change from system to system and from year to year, we choose to acquire a new battle-management system as a new development. Almost exclusively changes are to the sensors used to collect information for the warfighters, the weapons used to engage threat targets, and the rules of engagement (ROEs) established in both the planning and the C2 functions. Specific features within the battle-management software will change over time (e.g., discrimination algorithms, correlation algorithms, feature-aided tracking); however, one can isolate those features in components that can be interchanged when developers are prepared to introduce new components (e.g., new types of weapon systems) into the battle-management software.

3. A NEW PARADIGM

We believe that it is possible to develop a globally distributed, real-time software-intensive battle-management system that exhibits highly predictable system-software behavior, in which the system receives sensor information from land, sea, air, and space, and commits land-, sea-, air-, and space-based weapons to fire at identified targets. Furthermore, we believe that it is possible to employ linear temporal logic and model checking to a globally distributed, real-time battle-management system to aid in the realization of desired system behavior to include the weapons-commit logic. We argue that the major benefits of our paradigm are that it will provide the following:

- An engineering-based approach for developing battle-management kernels (BMK) for missile-defense and other types of real-time systems used by combatant commands.

- Acquisition organizations with a means for developing real-time software-intensive distributed systems that exhibit a high degree of predictability of system behavior.

3.1 Battle Management Kernel

The BMK is intended to serve as "glueware" between software applications unique to each battle-management domain, and the sensors, C2 systems, and weapon systems in that battle-management domain.[1] That is, the BMK will execute the five kill-chain functions by calling upon various components for computation, as described in [3].

Rather than point-to-point interfaces, we will develop type interfaces that define the behavior of each interface and the required specifications to realize each interface. Further, we will develop and maintain the interfaces as separate configurable items to preserve the identity of the interface and to minimize the opportunity for multiple versions of the interface. Additionally, it is important that the interface advertise its operations, while not specifying implementations of its operations [5].

For ease of integration and maintainability, we will develop software components for the features that typically experience the majority of changes. In theory, developers can realize a component-based framework with less effort and without unwieldy upgrade cycles as compared to fully integrated, monolithic software solutions. Additionally, a component-based framework allows for tailoring of the framework to address specific user needs [13]. For this paper, we identify software components that include enforcing rules of engagement, conducting discrimination and correlation, performing feature-aided tracking, and estimating launch, impact, and intercept points.

A BMK is similar in purpose to an operating system (OS) kernel in that both kernels manage resources shared by competing entities. In the case of an OS kernel, the competing entities are computer processes vying for processor and memory resources. In the case of a BMK, the competing entities are all of the components of the system-of-systems that comprise the battle-management system, such as the C2 and weapon systems. The components in the kernel are expected to be stable compared to the other components in the system-of-systems. For instance, device drivers tend to be updated frequently and therefore in principle should not be included in the OS kernel. If they are included (i.e., the case of a monolithic kernel), and even worse, tightly coupled to OS management functions, then it becomes challenging to make modifications to the kernel that do not affect other parts of the kernel. We would like to apply this same reasoning to BMK in order to simplify the design and maintenance of the kernels.

---

[1] We use the term "glueware" rather than "middleware" because we intend for the BMK to serve more that just as a software layer that provides both a programming abstraction and transparency (of the underlying heterogeneous systems).

We also draw a parallel between BMK and safety kernels. The functions to be included in a safety kernel are those that must be performed to maintain a safe system state or bring a system back into a safe state after the occurrence of a safety-critical event. No other functions may be included in a safety kernel. An automated train protection (ATP) system is an example of a safety kernel. Such kernels are well documented, validated, and verified before being considered for certification and accreditation. We view battle-management kernels in a similar light: they must work as advertised because the ability of the entire system-of-systems to be able to conduct warfare in the BMD battlespace is dependent on the BMK.

3.2 Capability-based Acquisition Process

In the context of DoD capability-based acquisition, the government specifies the capabilities for the system that are needed by the warfighter. Government contractors specify and refine the capabilities into system requirements, architectures, designs, and other system artifacts. In [2], we demonstrate how the Unified Modeling Language (UML) can be used to refine a system-of-systems. In this paper we extend our earlier investigation to include the explicit treatment of linear temporal logic for developing the BMK functional specifications and verifying the specifications using model checking.

In this approach, one starts by developing a framework that contains the proposed BMK along with the battle-management software components that will experience the most change during the acquisition life cycle of a battle-management system. We envision software engineers developing the BMK as a real-time set of system functionality that addresses its use by warfighters, starting from a high-level statement of capabilities and refining these statements into successively lower levels of system artifacts. We define the BMK to be the software that contains the basic functions of battle management that will remain stable over time. Derived from the kill chain [3], these basic battle-management functions are called tasks, and will manage the use of the system's computing resources to ensure that all time-critical, battle-management events are processed as efficiently as possible.

As the initial step to the BMK development, we are performing a domain analysis of the battle-management functions. This involves deriving warfighter usage requirements from battle-management use cases. We are refining the use cases concurrently with the development of sequence diagrams—these are used to capture messaging requirements among the derived classes from the use cases. We are using the aforementioned artifacts as the basis for creating state diagrams of the BMK, with the aim of identifying the desired battle-management behavior. The final step of the domain analysis will be to identify and verify assumptions on battle-management operations, in support of developing BMK specifications.

From the iterative review and refinement of the aforementioned artifacts, we will develop detailed specifications that focus on defining BMK behavior and achieving battle-management goals.

4. FORMAL APPROACH TO BMK DESIGN

Our approach includes the verification of the functional specifications with the use of model-checking tools to determine the degree of predictability of system behavior with respect to state transitions and tolerance to battlespace environmental variables. The verification will focus on ensuring that the BMK can both meet the specifications and exhibit the desired behavior. We will design test oracles that contain the full range of battle-management variables that are both within and outside the expected range of operational values for ballistic missile defense.

4.1 Temporal Logic

We will construct a set of specifications using temporal logic that will serve as a model of the BMK. The goal is to achieve a greater degree of clarity and focus in the specification of the desired BMK behavior as compared to that obtained from the traditional method of simply listing the system requirements.

We will develop a sufficient amount of information to automatically produce test cases for the implementation. Otherwise, we run the risk of developing so-called "cartoon models" that are only useful for drafting and refining potential solutions. We will develop test-ready models of the BMK. According to Binder [1], in order to be testable, a model should contain all the features of the system-under test (in the present context, the BMK), preserve sufficient detail that is critical for discovering faults, and faithfully represents the essential states, actions, and transitions in the state diagram. If the BMK model is to be useful for this effort and in future acquisition efforts, it must exhibit the following properties outlined in [12]: appropriate level of abstraction, high degree of understandability, high measure of accuracy, and high level of predictiveness.

We must ensure that the BMK model faithfully represents the behavior of battle-management operations. We must ensure that the BMK states are reached appropriately and the transition triggers are reflective of the projected BMD battlespace. For example, the model must transition from the state in which tracking occurs to the state in which a weapon assignment occurs each and every time the model is presented with the appropriate transition events. Just as important, the model must not transition for events other than what was designed for the BMK.

We are using temporal logic to define assertions for the BMK specifications. We anticipate that these assertions will yield specifications that are verifiably consistent and accurate, and in turn, verifiably predictable behavior of the BMK.

It is our experience that the vast majority of engineers involved with acquisition of software-intensive systems are not familiar with software formalisms. Additionally, we assert that few of the many system engineers in acquisition could follow temporal logic without some level of instruction. As such, software engineers may choose to minimize the use of typical temporal logic symbols and attempt to develop the specifications in as close to natural language as possible while still maintaining the degree of rigor that temporal logic lends to specification development.[2]

This approach is necessary to gain buy-in from system engineers and engineering managers. Acquisition efforts require

[2] Other alternatives to presenting system engineers and program managers with mathematical formulas include but are not limited to tabular and graphical representations, along with simulation and animation (e.g., animate the transitions among states in a statechart as a simulation of a system progresses).

significant commitments of human and financial capital. Introducing new acquisition methods to replace that which is familiar and comfortable is generally viewed as risky and foolish. Proposed changes must be readily evident to system engineers and engineering managers, or the proposed changes will not be adopted. As a partial example of our approach, consider the assignment of a weapon to a tracked object:

User Goal: Assign a Weapon to a Tracked Object
Narrative: The BMK must assign a weapon to engage a threat object before it impacts or detonates over a pre-designated defended area. The BMK must determine whether the tracked object is a ballistic-missile threat. The BMK must determine whether the predicted impact point is within the defended area as defined by military planners. The BMK must determine which weapons are available. The BMK must determine which weapon(s) can engage the tracked object. The BMK must assign the appropriate weapon to prosecute the engagement of the tracked object.

The logic to assign a weapon to a tracked object is as follows:

Weapon assigned to track object is true iff:
(Tracked object is a ballistic-missile threat) ∧
(Predicted impact point is within defended area) ∧
(Weapon is available) ∧
(Weapon interceptor capability is adequate)

We would outline the specification as follows:

Variables:
Boolean: Weapon_Assigned
// Weapon assigned to tracked object is true
Boolean: Ballistic_Threat
// Tracked object is ballistic-missile threat
Boolean: IPP_Within_Defended_Area
// This statement is true if the predicted impact point lies inside the physical dimensions of the defended area.
Boolean: Weapon_Available
// True if one or weapons are capable of immediately launching an interceptor.
Boolean: Intercept_Point_Min_Within_Intercept_Range
// True if the minimum intercept point lies within the interceptor range volume.
Boolean: Unknown Track
// True if track object has yet to be identified as a ballistic-missile threat
Set: Tracked_Object
// Contains detected characteristics of a ballistic-missile threat
Multiset: Threat_Profile
// Contains sets of characteristics for known ballistic-missile threats
String: Tracked_Object_Status
// Identifies status of Tracked_Object. Will be Active, Killed, Hit, or Dropped
Integer: Unknown_Track_Life
// Time duration from detection to present time – expressed in seconds.
Boolean: IPP_Within_Defended_Area
// True if ballistic-missile threat IPP lies within defended area
Real: IPP_Latitude
// Latitude of IPP
Real: IPP_Longitude
// Longitude of IPP
Real: Defended_Area_Max_Latitude
// Maximum latitude value of defended area
Real: Defended_Area_Min_Latitude
// Minimum latitude value of defended area
Real: Defended_Area_Max_Longitude
// Maximum longitude value of defended area
Real: Defended_Area_Min_Longitude
// Minimum longitude value of defended area
Boolean: Weapon_Status_Operational
// True if weapon is operationally available to fight
Boolean: Weapon_Launcher_Armed
// True if weapon launcher is armed and ready to fire
Set: Min_Intercept_Point
// Minimum intercept point at which an intercept at points closer to defended area would result in negative consequences to the defended area. Expressed in longitude and latitude. Typed as a set.
Multiset: Interceptor_Range_Volume
// All points within the range of the interceptor. Expressed in longitude and latitude.

Assertions:
Always Weapon_Assigned ⇔ ((Ballistic_Threat) ∧ (IPP_Within_Defended_Area) ∧ (Weapon_Available) ∧ (Intercept_Point_Min_Within_Intercept_Range))

Always Ballistic_Threat ⇔ (Unknown_Track) Until
(Tracked_Object ∩ Threat_Profile) ∧
(Tracked_Object_Status = Active) ∧
(Unknown_Track_Life) < 60

Always IPP_Within_Defended_Area ⇔
(IPP_Latitude ≤
Defended_Area_Max_Latitude) ∧
(IPP_Latitude ≥ Defended_Area_Min_Latitude) ∧
(IPP_Longitude ≤ Defended_Area_Max_Longitude) ∧
(IPP_Longitude ≥ Defended_Area_Min_Longitude)

Always Weapon_Available ⇔
(Weapon_Health_Operational) ∧
(Weapon_Launcher_Armed)

Always Intercept_Point_Min_Within_Intercept_Range
⇔ (Min_Intercept_Point ∩ Interceptor_Range_Volume)

4.2 Model Checking

Software engineers should verify the functional specifications via model checking. We define model checking as the systematic approach for testing functional assertions and substantiating the desired system behavior in the model. Model checking is not a proof of correctness; instead, model checking involves creating functional models of a system and analyzing the model against the formal representations of the desired behavior [8].

For the BMK, we verify the functional specifications using an automated model-checking tool that can accept either developed specifications or UML statecharts as discussed in [7], and exercise the temporal-logic assertions over a number of time cycles. We identify inconsistencies and breaks in logic through the use of the model-checking tool. From the results of the model checking, we intend to correct our specifications and the artifacts from the domain analysis as required.

However, our use of model checking is constrained by the state-explosion problem: as the size of the state space exceeds the memory capacity of the automated tool to check every trace in the model [6]. Through abstraction of the BMK functions in our specifications, we employ the concept of symbolic model checking in which Boolean functions are employed to represent transition relations and sets of states, using, for instance, a compact representation of the state space (e.g., binary decision diagrams [4]), to simplify the BMK states by removing sub-trees and redundant edges on the BMK's Boolean decision tree. In other words, we transform the complex logic decisions at the bottom of the tree into simple Boolean statements so that we can capture the essence of the system behavior in the upper portions of the decision tree. By reducing the high number of lower-level logic statements that develop very specific solutions and have limited impact on the overall system behavior, we should be able to manage the state-explosion problem.

As an example of the state-explosion problem in terms of the BMK, consider the following assertion:

Always Intercept_Point_Min_Within_Intercept_Range
$\Leftrightarrow$ (*Min_Intercept_Point* ∩
Interceptor_Range_Volume)

Note that the number of points in *Interceptor_Range_Volume* could be large and that we are seeking to ensure that one specific point (*Min_Intercept_Point*) is within the set of points that define *Interceptor_Range_Volume*. Rather than use model checking to ensure that this condition is true, we could abstract the assertion to either a True or False for *Intercept_Point_Min_Within_Intercept_Range*. This will reduce the number of traces through the model to verify this assertion.

4.3 Missile Defense System Architectural Framework

We are developing a framework in which the BMK connects to software components used for calculations in battle management as well as the interfaces to external components of systems such as sensors, C2, and weapons. The objective of this framework is to show a design of a battle manager as an integration of various components rather than a single software application. Here, as in our past research [2], we consider weapon systems to be comprised of components rather than a single entity.

By treating the each software application and each software interface as components, we believe that acquisition organizations can develop battle managers with more efficiency, reduced development times, and higher quality than current state-of-the-practice methods; a similar view is expressed in [5]. The high-level architectural view for the BMK is depicted in Figure 1.

4.4 Prototype Validation

Software engineers should consider developing a prototype of the BMK framework and demonstrating its behavior, capabilities, and limitations. They should test this prototype to determine the degree of system-behavior predictability with respect to state transitions and tolerance to battlespace environmental variables.

While the demonstration is not intended to be an exhaustive test, it will offer a degree of robustness to accompany the capabilities of the BMK prototype; we use the term "robustness" here to mean the degree to which a system can tolerate both failures and faults. In other words, a robust system handles unexpected states in a manner that minimizes performance degradation, data corruption, and incorrect output.

We propose the following partial list of metrics be used as part of the BMK demonstration:

1. Maximum number of concurrent tracks
2. Percentage of processed tracks (birth to death) to total received tracks
3. Percentage of correlated tracks to total correlation opportunities
4. Percentage of discriminated tracks to total discrimination opportunities
5. Percentage of weapon/target assignments to total weapon/target pairing opportunities
6. Percentage of received weapon assignments to total weapon assignment opportunities
7. Percentage of launch authorizations to total weapon assignment opportunities
8. Percentage of re-engaged tracks to total re-engagement opportunities
9. Percentage of undesired state changes to total illegal and out-of-bounds inputs
10. Percentage of system crashes and system lockups to total illegal and out-of-bound inputs

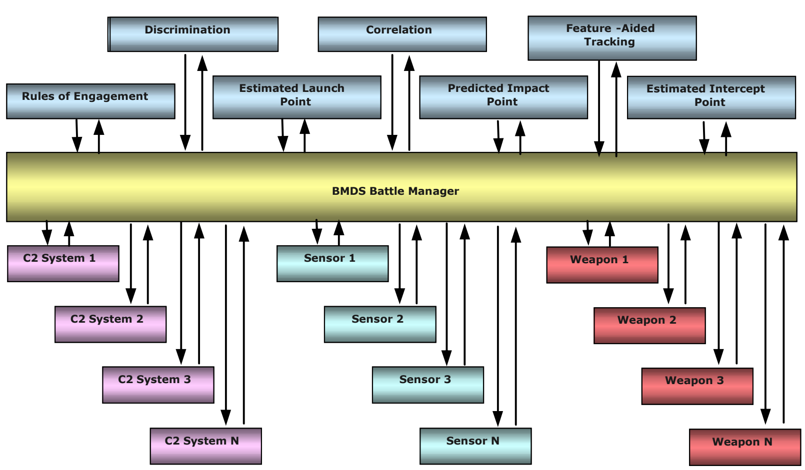

Figure 1. Missile Defense System Architectural Framework

5. DISCUSSION

In our approach, only the basic functions of battle management that will remain stable over time are to be placed in the kernel. The BMK will manage the use of system's computing resources to ensure that all time-critical, battle-management events are processed expeditiously. The development of the BMK begins with articulating the capabilities, followed by refining the capabilities into system requirements and properties, and finally into test- and verification-ready models (i.e., representations of the system-of-systems that are amendable to automated testing and verification). To avoid the state-explosion problem, software engineers should carefully model the fundamental behavior of the BMK rather than a comprehensive specification of the BMK.

Software engineers should consider developing the formal representation of the BMK by using temporal logic to describe the functional specifications of the BMK. They should verify the functional specifications with the use of a model-checking tool to determine the degree of system-behavior predictability with respect to state transitions and tolerance to battlespace environmental variables. By incorporating assertion error-handling schemes developed from the functional model and verified by the model-checking effort into the BMK, software engineers can develop embedded automatic test generation capabilities.

Software engineers should develop the required BMK interfaces as type interfaces that define the behavior of each interface and the required specifications to realize each interface. Desired timing-related behaviors of the system-of-systems should be represented in the interface definition rather than depending solely on messaging requirements; these interfaces need to be constructed for all BMD elements.

For ease of integration and maintainability, we should develop software components to encapsulate the features that typically experience the majority of changes.

6. CONCLUSION

It is our belief that software engineers can develop a BMK that addresses the five basic functions and fulfills basic warfighter usage requirements for a battle-management capability. Acquisition agencies within DoD can use the proposed BMK approach as a point of departure for developing systems-of-systems. However we should assess the utility of the BMK approach on real systems, including the BMDS, in order to confirm whether the approach both (i) contributes to the system-of-systems exhibiting high degrees of system-behavior predictability with respect to timing requirements and (ii) results in battle-management kernels that support plug-and-play of system components without violating the BMK timing requirements.

ACKNOWLEDGEMENTS

The research reported in this article was funded by a grant from the U.S. Missile Defense Agency. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotations thereon.

REFERENCES

[1] Binder, R. V. **Testing Object-Oriented Systems: Models, Patterns, and Tools**, Reading, Mass.: Addison-Wesley, June 2001.

[2] Caffall, D. S. and Michael, J. B. A new paradigm for requirements specification and analysis of system-of-systems. In Wirsing, M., Balsamo, S., and Knapp, A., eds**., Lecture Notes in Computer Science: Radical Innovations of Software and Systems Engin. in the Future**, Vol. 2941, Berlin: Springer-Verlag, 2004, pp. 108-121.

[3] Caffall, D.S. and Michael, J.B. **Developing Highly Predictable System Behavior in Real-Time Battle-Management Software**, Technical Report NPS-CS-03-006, Naval Postgraduate School, Monterey, Calif., Sept. 2003.

[4] Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. Progress on the state explosion problem in model checking. In Wilhelm, R., ed., **Lecture Notes in Computer Science: Informatics - 10 Years Back. 10 Years Ahead**, Vol. 2000, Berlin: Springer-Verlag, 2001, pp. 176-194.

[5] Crnkovic, I. and Larsson, M., eds. **Building Reliable Component-Based Software Systems**, Norwood, Mass.: Artech House, 2002.

[6] del Mar Gallardo, M., Martínez, J., Merino, P., and Pimentel, E. Abstract model checking and refinement of temporal logic in αSPIN. In **Proc. Third Int. Conf. on Application of Concurrency to System Design**, IEEE (Guimarães, Port., June 2003), pp. 245-246.

[7] Gnesi, S., Latella, D., and Massink, M. Model checking UML Statechart diagrams using JACK. In **Proc. Fourth Int. Symposium on High Assurance Systems Engin.**, IEEE, (Washington, D.C., Nov. 1999), pp 46-55.

[8] Lewis, G. A., Comella-Dorda, S., Gluch, D. P., Hudak, J., and Weinstock, C. **Model-Based Verification: Analysis Guidelines**, Technical Note CMU/SEI-2001-TN-028, Software Engineering Institute, Pittsburgh, Penn., Dec. 2001.

[9] Meyers, C. B., Feiler, P. H., Marz, T. In **Proc. Real-Time Systems Engin. Workshop**, Special Report CMU/SEI-2001-SR-022, Software Engineering Inst., Pittsburgh, Penn., Aug. 2001.

[10] Parnas, D. L. **Software Fundamentals: Collected Papers by David L. Parnas**, Reading, Mass.: Addison-Wesley, 2001.

[11] Paul, R. Rapid and adaptive end-to-end T&E of joint systems of systems. Presentation at the Fifteenth Annual Software Technology Conf., Salt Lake City, Ut., Apr. 2003.

[12] Selic, B. The pragmatics of model-driven development, **IEEE Software**, Sept./Oct. 2003, pp. 19-25.

[13] Szyperski, C. **Component Software: Beyond Object-Oriented Programming**, Reading, Mass.: Addison-Wesley, 2nd ed., 2002.

.